MATER: Mutually Aware Framework for Teleoperated-robot with Extended Reality

ZILIANG ZHANG, University of California, Riverside, USA
CONG LIU, University of California, Riverside, USA
HYOSEUNG KIM, University of California, Riverside, USA

Robot teleoperation with extended reality (XR teleoperation) enables intuitive interaction by mapping user motions to remote robots with real-time 3D feedback. However, existing systems suffer from large completion delays and trajectory deviations under prolonged network latency, rooted in their exclusive reliance on network communication and strict synchronous execution architecture. Moreover, network fluctuations destabilize teleoperation accuracy, while dynamic user motions amplify teleoperation errors.

We present MATER, an end-to-end XR teleoperation framework that introduces a mutually-aware architecture in which each side reconstructs its counterpart's delayed or missing state to decouple the execution from network dependency. MATER includes latency-adaptive input window and user motion gap interpolation techniques to handle unstable network communication. It also proposes motion-driven robot state rollback and robot trajectory coordination to handle complex motions. The key idea behind these techniques is to adapt on the fly by reshaping reconstruction and filling gaps as network conditions fluctuate, and by realigning states when motions become fast or complex. Together with lightweight local synchronization and bandwidth optimizations, these system-level advances make MATER resilient to both network and motion dynamics.

We implement MATER across three hardware settings, including simulated and physical robots, and evaluate it on 9,500 real-world teleoperation trials from the RoboSet dataset [1], covering single- and multi-step missions. Compared to state-of-the-art XR teleoperation frameworks, MATER reduces teleoperation error by up to 69.8% on WLAN and 73.1% on cellular networks with only 6.7% maximum runtime overhead. It also shortens mission completion time by up to 47.7%, enabling smoother teleoperation. A real-world case study on ten stationary and mobile missions further shows MATER achieves up to 37.7% faster completion while lowering average teleoperation error by up to 57.2%. MATER code is available at: https://github.com/rtenlab/mater

CCS Concepts: • **Computer systems organization** → *Robotics*; • **Human-centered computing** → *Virtual reality*; *Mixed / augmented reality*; • **Networks** → *Network performance evaluation.*

Additional Key Words and Phrases: Software Architecture, Extended Reality, Robotics, Teleoperation.

ACM Reference Format:
Ziliang Zhang, Cong Liu, and Hyoseung Kim. 2026. MATER: Mutually Aware Framework for Teleoperated-robot with Extended Reality. *Proc. ACM Meas. Anal. Comput. Syst.* 10, 2, Article 45 (June 2026), 23 pages. https://doi.org/10.1145/3805643

1 Introduction

Robot teleoperation via Extended Reality (XR teleoperation) emerges as a transformative technology in industrial automation, medical robotics, autonomous robot training, and humanoid robotics [2–11]. In XR teleoperation, an XR device uses onboard sensors to capture user motions and generate user poses, which are transmitted to a remote robot. The robot imitates these poses and sends back

Authors' Contact Information: Ziliang Zhang, University of California, Riverside, Riverside, CA, USA, zzhan357@ucr.edu; Cong Liu, University of California, Riverside, Riverside, CA, USA, congl@ucr.edu; Hyoseung Kim, University of California, Riverside, Riverside, CA, USA, hyoseung@ucr.edu.

ACM 2476-1249/2026/6-ART45
https://doi.org/10.1145/3805643

its own state and the surrounding environment. The XR device then fuses the latest user pose with the returned robot state and environment to render a 3D frame on a head-mounted display (HMD). By referencing both user and robot poses in this frame, the user adjusts next motion in real time, repeating the cycle until the mission is completed.

Today's XR teleoperation systems remain fragile, suffering from significant delays between user motion and robot feedback, primarily due to the network latency and the long control-feedback pipeline [2, 6, 7, 10, 12–16]. Unlike traditional teleoperation, immersive 3D rendering in XR teleoperation makes network delays highly perceptible as persistent discrepancies between the user pose and the robot pose in the displayed frame, known as teleoperation error. Continuing motion despite this error makes the robot drift from the user's intent, while pausing to wait slows mission completion. The root cause is *the synchronous execution architecture*: current systems require both sides to wait for network updates to maintain state consistency. This creates tight coupling, where the robot needs XR-to-robot messages to follow user poses, while the XR device requires robot-to-XR updates to render the robot state and environment. This dual reliance leaves the control-feedback loop highly vulnerable to even the modest network delay or fluctuation (**C1**), and highly sensitive to even the slightest increase in user motion speed or curvature (**C2**).

Contributions. We present **MATER**: **M**utually **A**ware **T**eleoperated-**E**xtended **R**eality, an end-to-end XR teleoperation framework that breaks the exclusive network reliance through bidirectional state reconstruction. MATER allows each side to locally reconstruct the other's delayed or missing information using local sensing and prior received information, decoupling robot control and XR visualization from round-trip communication. Therefore, each side maintains its own local state, enabling globally asynchronous execution. This architectural shift from the conventional, strict network-dependent XR teleoperation makes MATER naturally resilient to prolonged network delay.

Rather than relying on any single component, MATER's contribution lies in a coordinated system-level design that jointly addresses both challenges under network-induced latency, bandwidth variations, and motion-induced dynamics. Specifically, to address network fluctuation in (C1), MATER introduces a latency-adaptive input window technique on both sides that dynamically adjusts the input sizes based on observed latency, stabilizing reconstruction accuracy under unstable networks. On the robot side, a user motion gap interpolation component further mitigates faulty planner outputs when pose data is dropped or arrives out-of-order. To handle motion dynamics in (C2), a motion-driven robot state rollback reverts the reconstructed robot state according to the current user motion, while a robot trajectory coordination mechanism re-adjusts the robot's actual movement command based on reconstructed robot states and robot physical constraints, keeping robot trajectories aligned with XR displayed frames.

In addition, while MATER's mutually-aware paradigm naturally leads to an asynchronous execution model between XR and robot globally, MATER employs a synchronous execution model for local components on each device to eliminate instability from resource contention, where competing GPU kernels and CPU threads cause jitters, skipped executions, and stale data propagation. Finally, to address bandwidth variation that risks further pose drops, MATER implements bandwidth-adaptive data flow scaling, which prioritizes pose transmission by reducing the data that has little impact on the current cycle of teleoperation. The key idea is to adapt on the fly: MATER reshapes the system parameters to reconstruct motions when networks fluctuate, and realigns states and trajectories when user motions become faster or more complex, providing resilience with minimal runtime overhead.

We evaluate the MATER framework on three hardware settings over five motion patterns derived from 9,500 real robot teleoperation trials in the RoboSet dataset [1], a widely adopted dataset for autonomous robot training that covers 54 single- and multi-step missions. We experiment with four different network environments covering ubiquitous WLAN and cellular networks. Compared

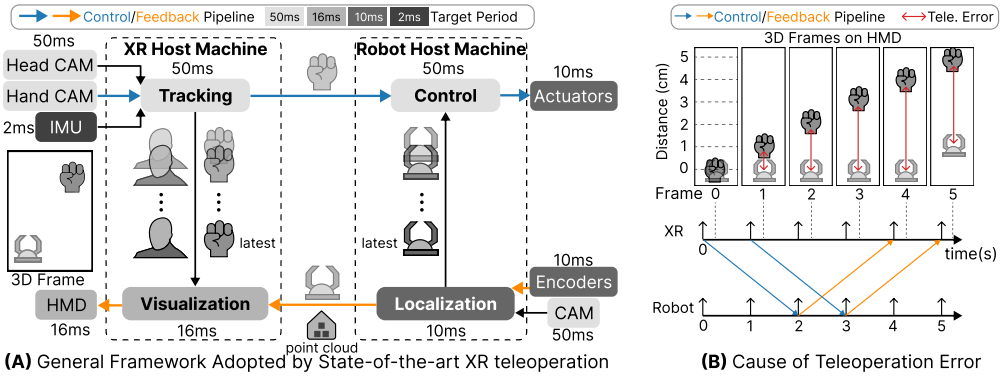


Fig. 1. Existing XR teleoperation framework and teleoperation error.

to the state-of-the-art XR teleoperation framework [13], MATER reduces the teleoperation error by up to 69.8% in WLAN and 73.1% in cellular networks, while shortening the mission completion time by up to 47.7% on a real robot manipulator [17]. We further conduct a real-world case study with 10 missions, including both stationary and mobile scenarios. In these trials, MATER achieves up to 37.7% faster mission completion time while maintaining on average 25.4% lower teleoperation error in stationary missions and 57.2% lower error in mobile missions.

2 Background and Challenges

2.1 Conventional XR Teleoperation

Conventional XR teleoperation follows an egocentric control model in which the user and robot share the same Cartesian coordinate system originating from the user center, effectively superimposing the user onto the robot for direct pose mapping [7, 13]. Existing XR teleoperation operates through *strictly synchronous pipelines*: the robot cannot move without receiving user poses from the XR device, and the XR device cannot update 3D frames without the robot's feedback. Consequently, every user action requires a complete round-trip communication, creating a fundamental dependency on the network.

Fig. 1(A) illustrates that a user performs a hand sweeping mission using the conventional software architecture that are widely adopted in state-of-the-art XR teleoperation [2, 7, 10, 13, 18, 19]. Initially, the Head CAM (camera) and Inertial Measurement Unit (IMU) capture the surrounding environment and the user's head inertial data, and the Hand CAM captures the user's hand. These sensor data are fed into a **control pipeline** (blue lines in the figure), which comprises *tracking* and *control* modules on the XR and robot, respectively. The tracking module constructs the *user pose* information by calculating the user's head movement from Head CAM and IMU data and the user's hand movement from Hand CAM data.[1] Then, the control module on the robot receives the user pose from the XR and generates control commands to move the robot's end-effector accordingly.

Since the user relies on 3D frames to observe the remote robot's pose and perform subsequent motions, the **feedback pipeline** (orange lines) comprises *localization* and *visualization* modules on the robot and XR, respectively. The localization module produces robot pose for the end-effector,[2] and point cloud data for surrounding objects in the environment [2, 13]. Then, the visualization module on the XR side combines the latest user pose (captured and processed in the latest sampling

[1]The user pose may include extra information such as hand gestures and point directions for sophisticated maneuvers.
[2]The robot may provide additional feedback information, such as gripper intensity and direction [6, 10, 20, 21].

period from the tracking module) with the received robot pose and point cloud to render 3D frames (typically at 60 FPS).

2.2 Performance Metrics

In conventional XR teleoperation, network delays play a pivotal role in creating temporal misalignment between user and robot poses displayed in the same 3D frame. For example, in Fig. 1(B), suppose that a user moves hand upward by five centimeters at a constant speed of one centimeter per second to control a remote robot. In each 3D frame, the hand symbol denotes the user pose generated on the XR side, while the robot manipulator symbol denotes the latest robot pose received from the remote robot. All XR and robot components operate at a one-second period, and one-way communication latency T_{onetrip} is two seconds. In the 3D frames from one to four seconds, the user can observe the misalignment of the user pose and robot pose as the robot pose is always received T_{onetrip} behind the user pose generation. To quantify the user-perceived spatial error in XR teleoperation due to the temporal misalignment explained above, we define the following metric:

Def. 1. Teleoperation Error. Teleoperation error E^{tele} is defined as the Euclidean distance between the user pose and the robot pose within the same 3D frame displayed to the user on HMD. Given the same 3D frame with the latest user pose u_{t_s} sampled at t_s by the the XR's Hand CAM and the latest robot pose sampled at t_r by the robot's encoder, the teleoperation error E^{tele} at current time t when seeing the 3D frame, E^{tele}_t, is given by:

$$E^{tele}_t = ||u_{t_s} - r_{t_r}||$$

This metric captures humanly perceivable misalignment that affects XR teleoperation quality, regardless of whether it arises from network latency and processing/sensing delays.

Teleoperation error affects not only spatial accuracy but also how long a user takes to finish a specific mission. Users often pause when the distance between the current user pose and the displayed robot pose exceeds a perceptible threshold in the 3D frame, and resume only after the displayed robot catches up. Without these pauses, teleoperation error can accumulate, causing the remote robot trajectory to deviate significantly from the user's intended motion [2]. As shown in Fig. 1(B), after the user reaches five centimeters up at five seconds, the displayed robot still reflects an earlier pose and does not catch up until four seconds later. This leads to the user to wait, directly increasing mission completion time. Measuring such waiting times individually is impractical because they are user-driven and occur irregularly. Therefore, we capture their impact through the total time required to complete a mission.

We define a *mission* as a chronologically ordered sequence of user motions that starts from and ends at a complete stop. For mission i, we denote its j-th user motion as $M_{i,j}$, with $j \in \{1, \ldots, J_i\}$. Each motion elapses exactly one period of XR tracking because the user pose is sent at the end of each XR tracking job. The mission begins with $t^u_{i,1}$ at the start of $M_{i,1}$, when the user initiates the first motion captured by the Hand CAM. It ends when both the remote robot at time t^r_{i,J_i} and the displayed robot at time t^d_{i,J_i} reach the last user motion $M_{i,j}$. We can define the following time metrics to complete the mission:

Def. 2. Mission Completion Time. Mission completion time is the duration from the start of the first user motion to the time when both the remote robot and the displayed robot have reached the user pose in the last user motion before returning to a complete stop. Hence, for mission i, mission completion time T^{mission}_i is given by:

$$T^{\text{mission}}_i = \max\left(t^r_{i,J_i},\ t^d_{i,J_i}\right) - t^u_{i,1}$$

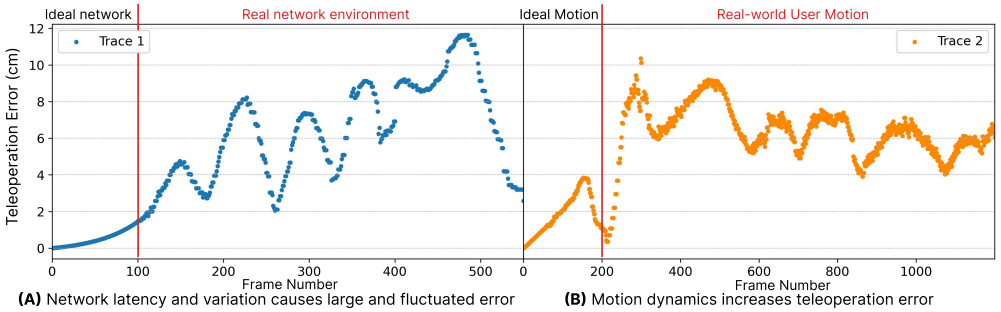


Fig. 2. Real-world XR teleoperation error traces

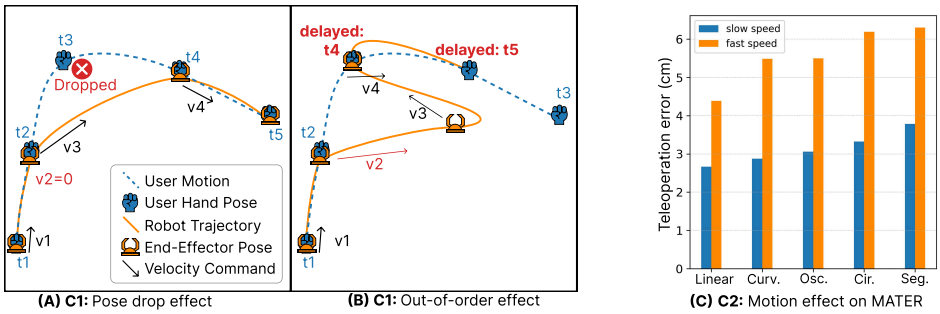
We take the maximum because either side can finish later. In practice, $t^r_{i,J_i} > t^d_{i,J_i}$ may happen due to robot actuation delays that may finish later than XR visualization. For a mission i with J_i motions, where the j-th motion has a one-way latency of $T^j_{onetrip}$ and a sensing period of T_{sensing} for XR tracking, T^{mission}_i in conventional XR teleoperation is lower-bounded by $(\sum_{i=1}^{J_i} T^j_{onetrip}) - T_{sensing}$. The lower bound holds because conventional XR teleoperation requires at least one one-way communication latency per motion to exchange the information from both sides. The subtraction of T_{sensing} accounts for the final visualization period, during which the user is already stationary, and the XR display can be updated without additional XR tracking execution.

2.3 Challenges

As shown in §2.1, unlike other distributed interactive system [22–26], conventional XR teleoperation uses a tightly coupled control and feedback loop that enforces a single synchronous state shared and executed by both sides, creating an unavoidable dependency on network communication. Any increase in network latency propagates through the pipeline, inflating teleoperation error and prolonging mission completion time [2, 27, 28]. This motivates removing the strict coupling by adopting a software architecture that allows each side to run on its own timeline with a separate state. However, such an architectural shift is difficult to realize because two fundamental challenges, C1 and C2, are inherent to practical XR teleoperation scenarios.

C1: Network-dependent Teleoperation Performance. To understand the overall performance of the XR teleoperation systems in a realistic network environment, we first run a state-of-the-art ROS Sharp framework [29] that is built based on the ROS Reality [13] and record its teleoperation error during the trial. We experiment with one pre-recorded user motion trace from Robotset dataset [1] that maintains constant speed and direction for motion while changing the network condition from an ideal one with low-latency, no fluctuation and bandwidth limitation to an actual network that has high latency, random packets drops and out-of-order arrival due to network fluctuations (see §5 for details on our experimental setup). As shown in Fig. 2(A), during the ideal network condition, the teleoperation error is sustainedly below 2 cm, while in the realistic network, it increases to over 10 cm and cannot fall below 2 cm at any point.

As observed in real network scenarios, longer network latency leads to a large teleoperation error because the XR visualization receives the robot pose from a longer time ago. Apart from the large network latency, pose drops and out-of-order arrivals are also major sources of degradation, caused by bandwidth and latency fluctuations. These errors lead to persistent teleoperation inaccuracies and robot trajectories deviating from the intended user motion. Fig. 3(A) illustrates the effect of pose drops, when five sequentially timestamped poses are sent to the robot, but t_3 fails to reach



(A) C1: Pose drop effect **(B) C1:** Out-of-order effect **(C) C2:** Motion effect on MATER

Fig. 3. Major challenges in XR-teleoperation systems

the robot. The robot controller generates commands that skip directly from t_2 to t_4, producing abrupt motion and irrecoverable deviation. Fig. 3(B) shows out-of-order arrival, where the robot prematurely executes t_3 before receiving the correct ones. Then robot controller pursues the delayed poses t_4 and t_5 when they arrive after, further distorting the trajectory. Enforcing strict sequential order with timestamps could avoid this, but at the cost of longer pauses and extended task mission completion time.

C2: Motion-driven Teleoperation Accuracy To examine the effect of motion dynamics on the existing teleoperation framework, we vary both the speed and curvature of the user motion while controlling the network condition. We first study the effect of speed using real user traces that begin with slow, constant-speed motion and then increase speed by up to five times in a random pattern after 200 frames. We evaluate teleoperation accuracy using teleoperation error, where a smaller value indicates a shorter distance between the displayed robot pose and the remote robot pose given a fixed network environment. As shown in Fig. 2(B), higher motion speeds increase teleoperation error, because the distance between the displayed robot and the remote robot naturally grows when user speed increases, inflating the entire teleoperation error in effect.

We then quantify the effect of curvature by categorizing the input user motion into five patterns with increasing curvature: linear, curved, oscillatory, circular, and segmented. Fig. 3(C) shows that higher curvature also increases teleoperation error. This occurs because higher-curvature motions amplify the impact of delay. Existing frameworks simply render the latest received robot pose, so the displayed motion lacks the temporal cues needed for accurate human perception [30]. The user may continue executing sharper turns without noticing the deviation between the remote and displayed robot. This accuracy degradation from both speed and curvature underscores the need to adapt teleoperation to complex or rapidly changing motion patterns, since unhandled motion dynamics can accumulate trajectory deviation and undermine mission completion.

3 MATER Architectural Design

Based on the observations in §2.3, we present **MATER**, which takes a mutually-aware paradigm to decouple the execution pipelines from the network dependency. Today's XR and robot systems already have sufficient sensing and computation resources to locally reconstruct the state of their counterpart, allowing each side to continue execution without relying solely on network communication. Building on this insight, MATER decouples both control and feedback pipelines from network dependency through the following major ideas: (i) Bidirectional reconstruction of time-shifted states, (ii) asynchronous global execution, and (iii) control/data plane separation.

Bidirectional Reconstruction of Time-Shifted States. Illustrated in Fig. 4(A) blue and orange arrows, MATER lets each side maintain a locally reconstructed, time-shifted state of its counterpart

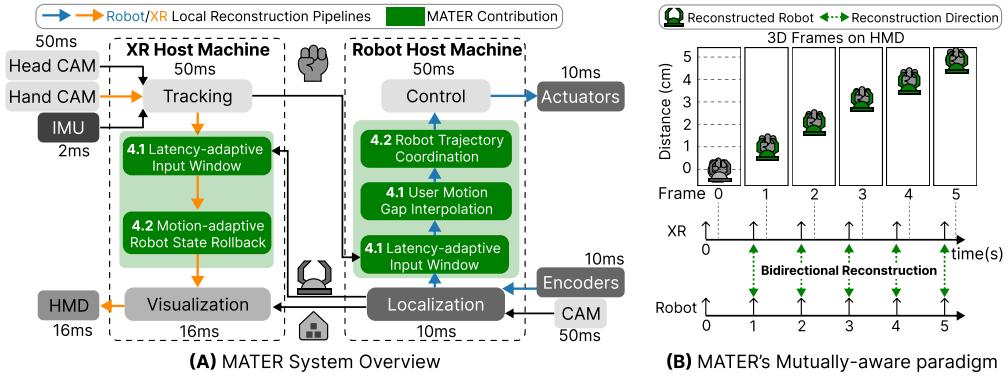


Fig. 4. MATER Software Architecture

by running local reconstruction pipelines. This differs from existing prediction-based techniques that only project a time-forwarded version of a single side's state [27, 28, 31, 32], as MATER allows both sides to reconstruct the missing information. The XR reconstructs the remote robot state based on the latest local user pose and a series of previously received robot poses, while the robot reconstructs user motion with the latest local robot pose and a series of received user poses. MATER's architecture does not restrict the choice of reconstruction algorithm; Time-forwarding [27], Extended Kalman Filter (EKF) [33], and Multilayer Perceptron [34] can all serve as the reconstruction model. Our implementation uses EKF by default due to its less intensive computation and low runtime overhead.[3] However, since reconstruction relies on poses transmitted over the network, its quality degrades as latency increases (C1). To maintain reconstruction quality under changing network conditions, we introduce a latency-adaptive input window technique (§4.1) that adjusts input sizes based on the runtime network latency.

Asynchronous Global and Synchronized Local Execution. The bidirectional reconstruction approach makes it possible to switch to asynchronous execution between the XR and robot. Unlike prior frameworks tied to a single, tightly synchronized state, MATER enables *asynchronous global states*, where XR visualization and robot control can progress independently and synchronize only when network messages arrive. With this asynchronous global execution, the challenges C1 (network-dependent performance) and C2 (motion-driven accuracy) can be addressed by optimizing the local reconstruction methods separately without causing interference to the global execution. On the XR side, a motion-adaptive robot state rollback selects the reconstructed robot state based on current motion to address C2 (§4.2). On the robot side, MATER implements a user motion gap interpolation technique that generates missing or delayed user poses during network communication to address C1 (§4.1). It then proposes a robot trajectory coordination technique to adjust the control command based on the poses displayed to the user and the physical constraints of the robot to address C2 (§4.2). While MATER enables asynchronous execution between XR and robot, it enforces the synchronous execution of local components on each side to avoid contention on local computation resources that causes runtime delay (§4.3).

Control/Data Plane Separation. The majority of existing XR teleoperation frameworks [2, 6, 10, 13, 18] use Data Distributed Services (DDS) for communication due to its compatibility with the Robotic Operating System (ROS). While this is fine for XR-to-robot communication as the

[3]This reconstruction is intended to bridge only short-term gaps in pose delivery, not to lower the transmission rate or replace continuous pose exchange, since without regular ground-truth correction, the locally reconstructed state will quickly accumulate teleoperation error.

XR sends only small payloads composed of user poses, robot-to-XR communication becomes an issue because payloads are much larger due to point cloud data, and DDS introduces noticeable serialization overhead for large payloads [35]. Motivated by this, MATER separates control (pose information) and data plane (point cloud). Control-critical pose information uses DDS for reliable transmission, and due to its low overhead with small payloads. Point cloud data for visualization uses UDP, allowing it to be dropped or delayed without affecting control accuracy. To minimize the human-perceivable effect from the point cloud dropping, MATER includes a bandwidth-adaptive data flow scaling technique to prioritize the drop of interior points based on current bandwidth to hold structural integrity(§4.3). Through these methods, both sides can maintain control correctness even when visualization data is compromised.

As shown in Fig. 4(B), under the same scenario as in Fig. 1(B), each side reconstructs the counterpart's delayed information during teleoperation using MATER. In each 3D frame, the green manipulator symbol denotes the XR-side reconstructed robot pose displayed to user. As a result, the user observes a robot motion that stays closely aligned with the hand motion, reducing visible teleoperation error and allowing the mission to be completed within five seconds.

4 Addressing Unique XR Teleoperation Challenges

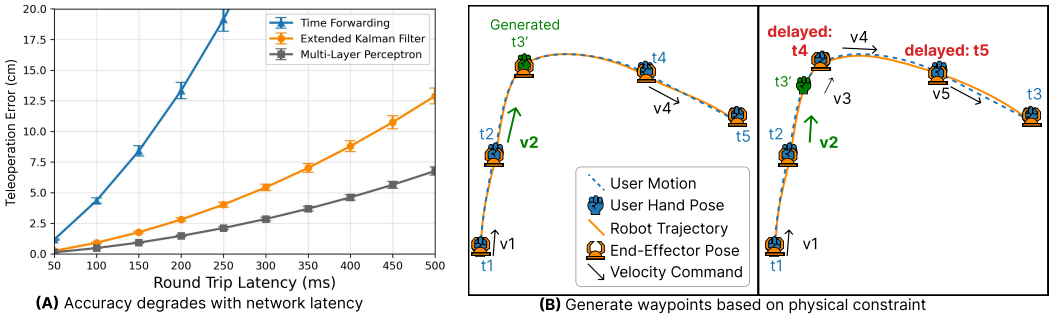
MATER's mutually-aware architecture, in which both the XR and robot independently reconstruct each other's state, creates new opportunities to address the challenges identified in §2.3. Because each side now maintains its own reconstructed view of the other, optimizations can be applied locally without requiring explicit cooperation over the network. This section presents techniques that exploit this architectural property to handle network fluctuations (C1) and motion dynamics (C2). §4.1 addresses C1 by improving reconstruction quality on both sides. §4.2 addresses C2 by adapting each side's reconstruction to motion complexity. Finally, §4.3 presents system-level optimizations that ensure the bidirectional reconstruction operates efficiently and reliably.

4.1 Handling Network Fluctuations

Latency-adaptive Input Window. MATER's bidirectional reconstruction requires both the XR and robot to maintain pose histories received from their counterpart. The quality of reconstruction therefore depends on how these histories are managed. Existing prediction-based methods implicitly assume that input samples are evenly spaced in time, where the received robot poses arrive periodically with a fixed interval [27, 28, 31]. Under this assumption, models can treat the pose sequence as uniformly sampled, but in practice, network delays cause variable spacing, drops, or out-of-order arrivals. As shown in Fig.5(A), such irregularities lead to unstable and highly variable accuracy for state-of-the-art algorithms[27, 33, 34], even when the underlying motion is smooth.

To address this, MATER adopts a *latency-adaptive* input strategy: rather than using a fixed-size, evenly spaced history, MATER determines the effective input window length from the most recently profiled round-trip latency and the robot encoder period. MATER caps the window length so that the input reflects the *current* network regime instead of mixing in much older samples collected under different delay conditions, and to prevent unbounded window growth under sustained high latency. Importantly, network fluctuations affect both XR and robot components, so latency-adaptive input window benefits both: the XR side stabilizes reconstruction for rendering, while the robot side receives more consistent state estimates to guide planning.

We adapt the input series length to the most recent profiled round-trip latency Δt_t. Let τ denote the robot encoder period. Here, t indexes the robot-side control step (occurring every τ seconds), and Δt_t is the profiled round-trip latency at step t. To cover each cycle of profiled round-trip latency, MATER selects the input window length k_t as the number of encoder intervals spanned by Δt_t, and then adjusts it to the user-selected range $[k_{\min}, k_{\max}]$ to avoid pathological sizes under extremely



(A) Accuracy degrades with network latency

(B) Generate waypoints based on physical constraint

Fig. 5. Network latency and fluctuations effect on MATER

low or high delay:

$$k_t^\star = \left\lceil \frac{\Delta t_t}{\tau} \right\rceil, \quad k_t = \min\left\{ \max\{k_{\min}, k_t^\star\}, k_{\max} \right\} \tag{1}$$

To reduce oscillation from minor latency jitters, we update the input window only when $|k_t - k_{t-1}| \geq 1$. At each control step t, we store the most recent k_t received robot poses together with their timestamps as $R_t = \left[(s_1, r_{s_1}), (s_2, r_{s_2}), \ldots, (s_{k_t}, r_{s_{k_t}})\right]$, where s_i are the received pose timestamps and may be irregular due to network fluctuations. Unlike fixed, uniformly spaced inputs, this representation preserves the true temporal spacing of the most recent poses in the input window, preventing the reconstructor from misinterpreting irregularly delayed samples as evenly spaced and thereby enabling stable state reconstruction under network fluctuation in **C1**.

Any reconstruction model can then operate on this time-aligned series, whether it uses simple extrapolation or more complex prediction models as discussed in Sec. 3. We can then represent the reconstruction in the following generalizable form:

$$\hat{r}_{t+\Delta t_t} = \mathcal{F}(R_t;\, \phi_t) \tag{2}$$

where $\mathcal{F}$ denotes a reconstruction with state R_t and additional parameters ϕ_t. Although we implement the $\mathcal{F}$ with EKF [33] for its low execution overhead, models with different complexities can be used based on practical deployment scenarios [27, 34].

Note that since MATER performs reconstruction on both sides, the latency-adaptive input window is applied symmetrically: robot state for visualization on the XR side, and user motion for control on the robot side. This is enabled by the mutually-aware architecture design, and ensures that network fluctuations are mitigated regardless of which direction is affected.

User Motion Gap Interpolation. Since MATER's robot-side reconstruction must generate control commands even when user poses are delayed or dropped, we interpolate missing user poses on the robot side to maintain a continuous, reliable target stream. This module operates at the same frequency as XR hand tracking and consumes each user pose update when it arrives, inserting it into a target queue; gaps occur when one or more intermediate pose updates are not received in time due to drops or reordering. After receiving the current robot pose r_t from robot localization, along with its velocity vector v_t (direction and speed), the reconstruction module invokes an online trajectory planner to fill in missing information.

Formally, let the target queue contain the next intended user pose u_{t+k} received from the XR side. Here, t is the current robot control step and $k \geq 1$ is the detected gap length between consecutive received targets in the pose stream. If certain intermediate poses are missing or delayed, we detect the gap from missing indices or timestamps in the incoming pose stream (e.g., a jump from u_t to u_{t+k}), indicating that $u_{t+1}, \ldots, u_{t+k-1}$ have not arrived in time. Then we interpolate waypoints

$\{w_{t+1}, w_{t+2}, \ldots, w_{t+k-1}\}$ on the curve between r_t and u_{t+k} using the current velocity estimate v_t:

$$w_{t+i} = r_t + \frac{i}{k}(u_{t+k} - r_t), \quad i = 1, \ldots, k-1 \tag{3}$$

These waypoints align with the physical constraints of robot kinematics, since both the step size and spacing are governed by the measured speed $\|v_t\|$ and direction $\frac{v_t}{\|v_t\|}$. As a result, even if pose u_{t+1} or u_{t+2} is missing, the interpolated trajectory maintains smooth continuity toward the next available pose, instead of forcing the robot to react only to sparse or delayed targets. This continuous target stream also helps keep robot execution aligned with the motion displayed on the XR side, reducing mismatch during network fluctuation in **C1**.

Fig. 5(B) left shows that with user motion reconstruction, the robot generates a missing waypoint t_3' from v_2, producing a smooth path through t_3', t_4, and t_5 that aligns with user motion. Similarly, Fig. 5(B) right shows a reconstructed t_3' that also mitigates the effect of delayed t_4 and t_5. Thus, waypoint generation from the latest pose and velocity restores trajectory continuity under network degradation.

4.2 Handling Motion-induced Reconstruction Error

Motion-adaptive Robot State Rollback. The XR side of MATER's mutually-aware architecture reconstructs the robot's current state to render 3D frames without waiting for network updates. However, reconstruction accuracy degrades under fast, high-curvature user motion (as shown with C2 in §2.3), and an inaccurate robot state may go unnoticed by the user. Since the XR now has its own reconstruction locally, it can actively manage reconstruction quality by selecting which reconstructed state to display. Specifically, when user motion changes abruptly or becomes too fast to reconstruct accurately between two pose transmissions, our mechanism rolls back to an earlier reconstructed state so that the 3D frame displays a robot position closer to the present, thereby encouraging the user to slow down.

When the motion becomes difficult to predict due to high speed or high curvature, the module may revert to a previous reconstruction state. We select the rollback amount with an online decision tree rollback policy:

$$j_t = \pi_\psi(z_t), \qquad j_t \in \{0, 1, \ldots, J_{\max}\} \tag{4}$$

where z_t collects recent motion, reconstruction, and network signals, and π_ψ is an online decision tree that maps z_t to a rollback amount j_t. If $j_t > 0$, the module restores the reconstruction state at time $t - j_t\tau$ and regenerates the reference using the newest observations; otherwise, it proceeds without rollback.

In practice, the rollback policy can be tuned through both the decision function π_ψ and the rollback bound $J_{\max}$. For smoother user motion, π_ψ is tuned to favor smaller rollback values j_t so that XR rendering remains as current as possible. For faster or higher-curvature motion, π_ψ is tuned more conservatively so that larger rollback values are selected more readily when reconstruction becomes less reliable. The bound $J_{\max}$ is chosen according to robot manipulator physical capability: robots with slower dynamics or stricter kinematic limits use a larger $J_{\max}$, while more agile manipulators can use a smaller $J_{\max}$.

Robot Trajectory Coordination. MATER's mutually-aware architecture creates two views of robot state: the XR's reconstruction (what the user sees) and the robot's local state. Without coordination, the robot may execute trajectories inconsistent with both sides, producing chasing behavior and larger teleoperation error. We achieve this coordination on the robot side with: (i) an inner loop that fuses XR reconstruction with robot-local trajectories to produce a consistent tracking goal, and (ii) an outer loop that refines this goal and the subsequent robot command based on the known physical constraints of the robot.

(i) Inner Loop Design. At time t, the XR side provides a time-stamped reconstruction snippet $\tilde{\mathcal{R}}_t = \{\tilde{r}_{t+\delta}, \tilde{r}_{t+2\delta}, \ldots, \tilde{r}_{t+H\delta}\}$, while the robot side has the current localized state (r_t, v_t) and the earliest unconsumed user target $u^\star$ from the queue $\mathcal{U}_t$.

$$g_t = (1 - \alpha_t)\, u^\star + \alpha_t\, \Pi_\Omega\big(\tilde{r}_{t+\Delta t}\big) \tag{5}$$

where Π_Ω projects any pose onto the robot's feasible set Ω given limits on speed, acceleration, and angular rate. The weight $\alpha_t \in [0, 1]$ depends on a confidence score c_t for the XR reconstruction and on link conditions,

$$\alpha_t \;=\; \sigma\Big(w^\top x_t\Big), \qquad x_t = \big[c_t,\ \Delta t,\ \text{jitter}_t,\ \|u_t - \tilde{r}_t\|,\ \|v_t\|\big] \tag{6}$$

with σ the logistic function. Here, α_t controls how strongly the robot follows the XR-side reconstruction versus the robot-local target. Since α_t is computed from x_t through the logistic function, its value naturally adapts to runtime conditions: when reconstruction is reliable and link conditions are stable, α_t becomes larger, causing the robot to follow the XR-side reconstruction more closely; when jitter, mismatch, or reconstruction residuals increase, α_t decreases, shifting reliance toward the robot-local target. Among the features in x_t, the confidence score c_t plays a key role. We compute c_t from recent reconstruction residuals observed when delayed ground truth arrives and from short input-window smoothness of $\tilde{\mathcal{R}}_t$.

The resulting coordination goal is then converted into the next robot reference under the robot's physical constraints before being streamed to the low-level controller. As new XR reconstructions and user poses arrive, the robot recomputes g_t and updates the reference, helping maintain alignment between robot execution and what the user sees on the HMD even under abrupt motion changes.

(ii) Outer Loop Design. Real robot mechanisms can substantially alter the robot trajectory that is reasonable in simulation [2, 10, 19]. To quantify this gap, we replay the oscillatory teleoperation pattern with MATER in simulation and on a physical robot teleoperated under 5GHz WLAN network condition. Fig. 6 shows 44.6% higher teleoperation error on the real robot, because actual physical constraints prevent it from achieving the aggressive trajectory achievable in simulation.

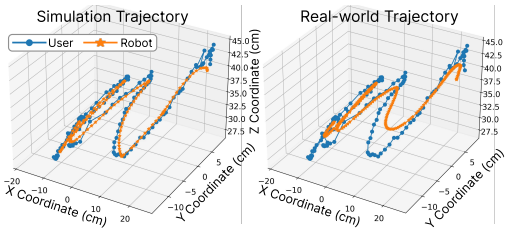


Fig. 6. Simulation vs. Real Robot running the same oscillatory motion.

To address this, the outer loop monitors tracking of the most recent reference and flags unreachable robot commands when the tracking residual exceeds the physical robot constraint, especially under abrupt motion, large Δt, or high jitter_t. When the command is flagged, it applies a lightweight planner-level PID correction [36] to the inner-loop generated waypoints before final robot command generation, while still enforcing the same feasibility checks and projection. It further adapts the motion limits used by the waypoint generator to ensure the resulting reference is physically trackable. Once tracking residuals fall below the tolerance, the outer loop returns to the nominal goal and limits.

4.3 Inter Component Coordination

Synchronous Local Execution. While MATER's mutually-aware architecture decouples global execution between XR and robot, it deliberately employs synchronous execution locally within each side. This is another key architectural shift from traditional XR teleoperation systems [13], which rely on asynchronous execution of local threads due to the pub-sub architecture of the underlying

Algorithm 1: Runtime Synchronization

Input : Tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, DAG $G = (V, E)$, periods $\{T_i\}$
Output: Ordered, synchronized execution per cycle

Setup: profile 99% exec times t_i; enforce $T_1 \geq \sum_{i=1}^{n-1} t_i$;
while *system active* **do**
  $Q \leftarrow \emptyset$; // ready queue for this cycle
  foreach $\tau_i \in \mathcal{T}$ **do**
    if *all predecessors of* τ_i *in* G *completed within* T_i **then**
      $Q \leftarrow Q \cup \{\tau_i\}$; // admissible this cycle
    end
  end
  while $Q \neq \emptyset$ **do**
    pick next τ_i by topological order of G;
    execute(τ_i) ; // pin critical tasks to reserved CPU cores; map GPU ops to a single stream
    mark τ_i as completed;
  end
  defer all non-admissible tasks to next cycle;
end

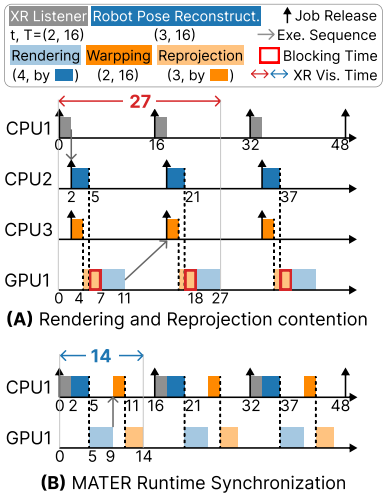


Fig. 7. Runtime Components Synchronization example.

DDS and ROS middleware. However, XR frameworks launch several GPU kernels and CPU threads for tracking, visualization, control, and localization concurrently. When these threads compete for shared GPU streams, CPU cores, or memory bandwidth, the resulting interference introduces jitters and skipped executions that propagate through the teleoperation loop. These runtime delays manifest as teleoperation error in a manner similar to network fluctuations: a skipped execution may resemble a dropped pose packet, while unexpected preemption can reorder outputs in ways that mimic out-of-order data arrival. Such intrinsic instability compounds extrinsic latency, creating additional drift between user and robot trajectories.

To address this, MATER introduces a synchronization-based scheduling strategy for local threads, which coordinates the execution of all components on each side. By enforcing consistent ordering and cycle boundaries, synchronization ensures that information generated in one cycle is fully consumed before the next begins. This prevents stale or premature data from leaking forward, eliminates jitter, and guarantees that downstream components always operate on coherent, up-to-date states. As a result, visualization on XR side and control on robot side remain consistent during execution, reducing teleoperation error without incurring significant runtime overhead.

We model each XR/robot component as a task that is either periodic with a fixed period or triggered by events (e.g., sensor arrivals, rendering completion). For a periodic task, each job is released at the beginning of its period and should complete before the period ends, so that outputs produced in one period are fully consumed within that period and do not leak into the next. Components share constrained local resources within XR or robot, including a limited number of CPU cores and a single GPU that is effectively serialized at the stream level. Conventional XR teleoperation frameworks typically map each component to a dedicated CPU thread to match the pub-sub execution model and to prevent tasks blocking on CPU level. For example, in Fig. 7(A), the three XR-side components run on three separate threads so that CPU computation on one component do not directly stall others, even though they may still contend for shared GPU execution.

MATER introduces a unified scheduling policy in Alg. 1 that synchronizes local GPU and CPU tasks within each XR and robot pipeline. During system setup, the scheduler enumerates all components $\tau_1, \tau_2, \ldots, \tau_n$ mapped to shared hardware resources, profiles their 99%-percentile

execution times $t_1, \ldots, t_n$ and records their target periods $T_1, \ldots, T_n$. These 99%-percentile execution times provide an upper-bounded basis for deciding whether an ordered set of jobs can reliably fit within one period. Therefore based on this profile, the scheduler builds a dependency graph $G = (V, E)$, where each node represents a component and edges encode data dependencies or resource conflicts.

Fig. 7(A) illustrates a case in XR visualization where rendering and reprojection contend for GPU resources without synchronization, leading to stale frames and an extended runtime of 27 units. In contrast, Fig. 7(B) shows the synchronized scheduler enforcing correct ordering, ensuring all tasks complete within one XR listener period. This reduces visualization runtime to 14 units, shortening the time to generate a 3D frame by 7 units compared to unsynchronized execution.

Bandwidth-adaptive Data Flow Scaling. As discussed in C1, limited bandwidth requires a mechanism to reduce sensor data size while preserving the visual quality needed for XR visualization. MATER adopts a separate control and data plane model, where the control plane dynamically scales sensor data flows to adapt to available bandwidth. Among these flows, point cloud data dominates network usage yet is indispensable for conveying both environmental context and object-level details around the remote robot. Simply extracting and transmitting object features would severely degrade situational awareness by discarding environmental cues. However, because human perception is especially sensitive to the structural integrity of point clouds [37], selectively maintaining edge points while reducing interior points can significantly lower data volume without noticeably compromising perceptual quality. Based on this insight, we design a point cloud scaling algorithm that dynamically removes interior points according to a scaling factor r, enabling bandwidth-aware adaptation with minimal quality loss.

During the setup phase, the available bandwidth B, the size of a single data point b, and the robot talker period T_{rt} are profiled and remain fixed at runtime. These parameters determine the maximum number of points that can be transmitted in a single point cloud as $N_{max} = \frac{B\,T_{rt}}{b}$. At runtime, upon scanning each point cloud from Robot CAM, MATER checks whether the number of points exceeds N_{max}. If it does, point cloud scaling is executed within the localization module at the start of the robot talker. The algorithm first applies Canny edge detection [38] to the latest point cloud to categorize the points into edge points, denoted as N_e, and interior points, denoted as N_{in}. To satisfy the bandwidth constraint, the scaling factor r is chosen to reduce only the interior points so that $N_e + r\,N_{in} \leq N_{max}$. Since the size of the point cloud and its number of points follow an inverse linear relationship, we define $r \in [1, 0)$ as follows: $r = \min\left\{1, \frac{N_{max} - N_e}{N_{in}}\right\}$.

5 Evaluation

5.1 Evaluation Setup

Hardware Setup. We employ three hardware configurations to encompass diverse deployment scenarios including pervasive simulation environment, physical robot setup, and practical embedded platforms. In the *Simulation* setup, the XR side framework operates on a PC equipped with an Intel i7-10700K CPU and an NVIDIA RTX 3060 GPU and displays 3D frame on a Northstar Next HMD [39]. A second PC with identical hardware hosts a simulated Kinova Gen3 robot manipulator using Gazebo [3] and transmits point-cloud scanned with a real Robot CAM. In the *PC-to-PC* configuration, we replace the simulated robot manipulator with a physical Kinova Gen3 robot manipulator while the remaining hardware stays the same. Lastly, *Xavier-to-Nano* deploys the XR-side framework on an NVIDIA AGX Xavier [40] and runs the robot framework on an NVIDIA Orin Nano [41] to test performance on embedded platforms commonly used in practical XR teleoperation scenarios. All sensors are set up as specified in Fig. 1.

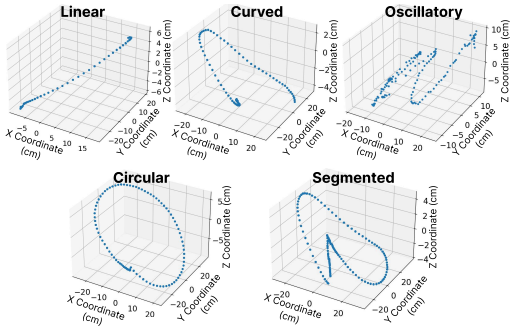


Fig. 8. Motion patterns example.

	Short Distance	Long Distance
WLAN 5GHz	77 ± 8 ms	236 ± 36 ms
WLAN 2.4GHz	83 ± 30 ms	270 ± 76 ms
Cellular 5G	87 ± 15 ms	282 ± 65 ms
Cellular 4G	118 ± 53 ms	302 ± 110 ms

$^{\Delta}$ Measured with `ping` during experiments.

Table 1. Network Roundtrip Delay and Fluctuation$^{\Delta}$

Motion Patterns. We analyzed 9,500 robot teleportation trials from the widely adopted RoboSet dataset [1] and identified five common hand-motion patterns consistently observed across these trials. These patterns include *Linear*, characterized by motion in a constant direction; *Curved* (Curv.), following a single directional change; *Oscillatory* (Osc.), exhibiting repeated back-and-forth movements; *Circular* (Cir.), involving continuous directional changes; and *Segmented* (Seg.), combining other patterns with intermittent pauses. For each identified pattern, we recorded 12 user hand movement videos as inputs for the XR Hand CAM. Fig. 8 provides illustrative examples for each motion pattern. To accommodate physical robot constraints, user hand movements were restricted to a maximum speed of 50 cm/s and a maximum reach of 80 cm. Using standardized motion patterns instead of entire mission videos from the original dataset allowed us to isolate the impact of joint jerks and external disturbances, preventing errors or delays unrelated to network and system factors. A detailed evaluation of real-world mission performance beyond these patterns will be presented in §5.4.

Network Configuration. We assessed system performance across four network configurations covering common fixed and mobile wireless communication scenarios. The WLAN configurations employed a TP-Link AC1750 WiFi router [42] operating at *WLAN 5GHz* with an 802.11ac interface and *WLAN 2.4GHz* with an 802.11n interface. The Cellular configurations utilized mobile hotspots created by a Samsung Galaxy S23 [43] running on *Cellular 5G* and *Cellular 4G* LTE networks. WLAN 5 GHz and Cellular 5G support up to 1300 megabits per second (Mbps) and 900 Mbps bandwidth respectively, both satisfying the requirement to exchange all information. In contrast, WLAN 2.4 GHz and Cellular 4G offer only 450 Mbps and 300 Mbps respectively, which are insufficient to transmit all information. XR and robot communicated through a cloud VPN server that can be migrated later. We adjusted the VPN server location to control latency and fluctuation (detailed locations omitted for double-blind reviews). During experiments, XR and robot positions remained fixed for stable network connections, and we measured round-trip latency and its fluctuation in standard deviation using ping commands, with results summarized in Tab. 1.

Baselines. We evaluated MATER against *ROS Reality* [13], a state-of-the-art XR teleoperation framework whose architecture was widely adopted in industry projects such as ROS Sharp [29]. To ensure a fair comparison, we modified ROS Reality to sample user poses from Hand CAM and replaced its original XR visualization with Godot [44] compatible with OpenXR [45]. Additionally, we designed two specific baselines to assess the effectiveness of the MATER software architecture and system optimizations: *MATER-NoOp*, which includes only the original robot-aware reconstructor and user-aware planner but lacks system optimizations described in §4.3; and *MATER*, representing

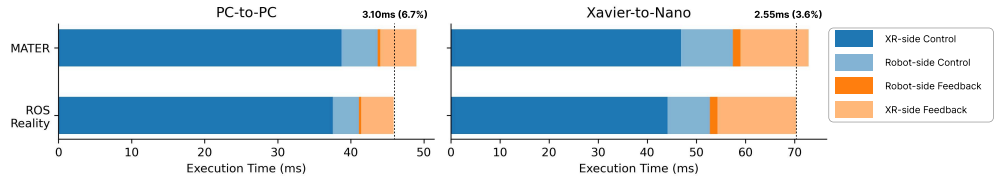


Fig. 9. MATER System Overhead.

our framework with all proposed features. All baseline configurations utilize the VR Gallery virtual environment [46] to control scene variability.

5.2 Controlled Motion Results

System Runtime Overhead. We evaluated both ROS Reality and MATER using linear motion as input and measured the execution time of each component during XR teleoperation [13, 30, 47]. We profiled the runtime of all components and grouped them into four phases: XR-side control, robot-side control, robot-side feedback, and XR-side feedback. Although MATER does not follow the same strict control-feedback loop as ROS Reality, we place XR-side reconstruction under XR-side feedback and robot-side reconstruction under robot-side control to enable a direct comparison. In contrast to MATER, ROS Reality only employs basic waypoint following described in §2.1 for robot control and viewport calculation [47] in XR visualization. Fig. 9 shows the runtime breakdown on both PC-to-PC and Xavier-to-Nano platforms as stacked bars. MATER incurs 6.7% overhead on PC-to-PC configuration and only 3.6% on Xavier-to-Nano configuration. Notably, the runtime for XR visualization on the Xavier-to-Nano is reduced by 13.4% owing to the local synchronization that mitigates GPU contention that causes blocking in ROS Reality.

Teleoperation Error. To obtain the teleoperation error in Def. 1, we recorded the Euclidean distance between the user pose and robot pose in each 3D frame. We use Euclidean distance as a basic measure of the user-perceived discrepancy in the 3D frame, since it directly captures the visible spatial mismatch between the current user pose and the displayed robot pose. However, because this metric alone does not fully capture the accumulated effect of reconstruction error during teleoperation, we will also report mission completion time. We then calculated the Root Mean Square (RMS) of these errors to better illustrate the magnitude differences across settings. Fig. 10, 11, 12 show the RMS teleoperation errors under the three hardware settings and all network configurations, with black error bars indicating the standard deviation within each motion pattern. Our ablation study isolates the effect of Inter-Component Coordination by comparing full MATER against MATER-NoOp, which retains bidirectional reconstruction on both XR and robot sides but removes the local coordination mechanisms. We do not evaluate XR-side-only or robot-side-only reconstruction as separate baselines, because MATER's asynchronous global execution requires reconstruction on both sides to run concurrently. Compared to ROS Reality, MATER-NoOp reduces RMS teleoperation error by up to 50.8% under WLAN and 54.5% under cellular networks in simulation settings. However, in 18 out of 24 settings, the standard deviation of teleoperation error in MATER-NoOp increases by up to 0.89 cm compared to ROS Reality, mainly due to the longer blocking time caused by extended execution time when XR reconstructs the robot pose. In contrast, MATER reduces RMS teleoperation error by up to 69.8% in WLAN and 73.1% in cellular networks, while its standard deviation only increases by up to 0.54 cm in 6 out of 24 settings, as the local synchronization mitigates contention by synchronizing the execution of all components in XR visualization.

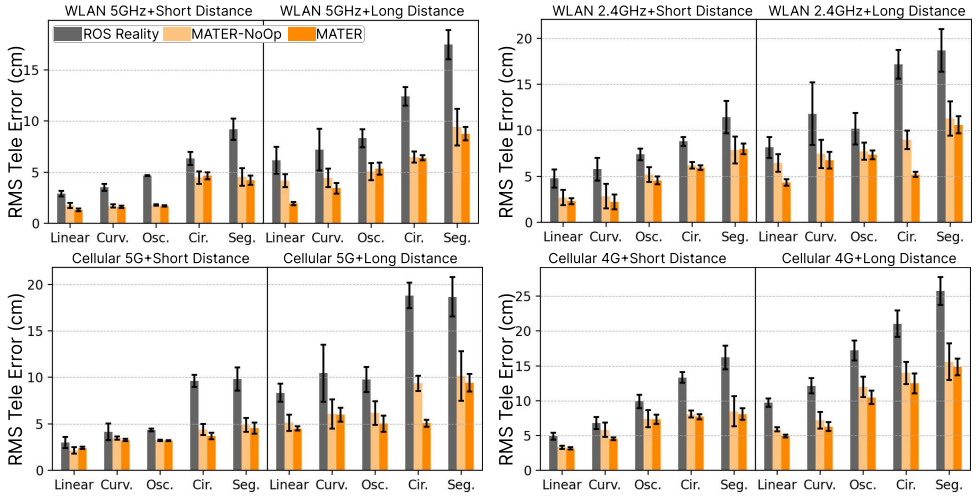


Fig. 10. Teleoperation Error comparison under Simulation. Root mean square teleoperation error (RMS Tele. Error) is used to better reflect the magnitude changes.

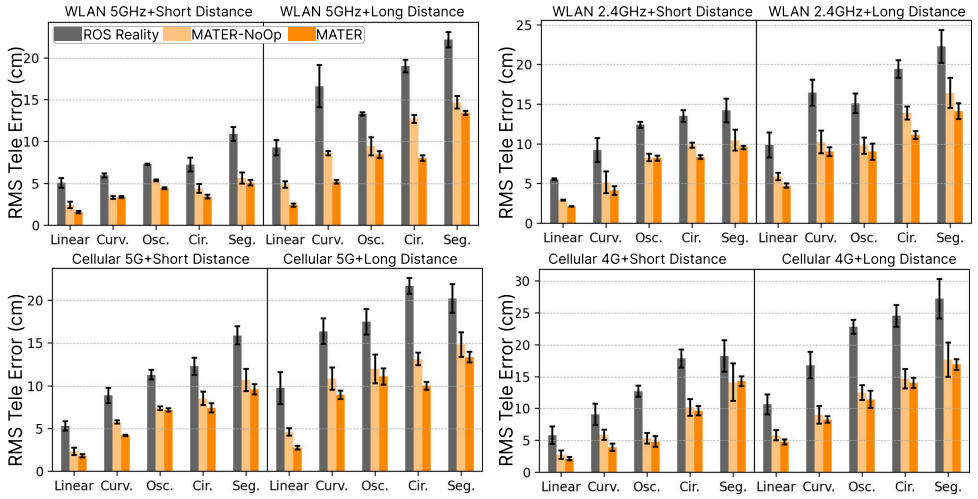


Fig. 11. Teleoperation Error comparison under PC-to-PC

Motion pattern complexity is a critical factor affecting teleoperation error. In experiments using a Xavier-to-Nano hardware setup on a Cellular 4G network, ROS Reality exhibited an increase in teleoperation error of up to 19.23 cm when transitioning from linear to segmented motion input as shown in Fig. 12. In contrast, MATER reduced this increase by 46.34% under the same conditions by reconstructing missing information locally for robot control and XR visualization. Nevertheless, due to limitations of the implemented Extended Kalman Filter and pure pursuit algorithms in the current framework, MATER still incurred a maximum 10.32 cm increase in teleoperation error when shifting from linear to segmented motion. These results suggest that more sophisticated approaches, such as a multilayer perceptron for robot state reconstruction, may more effectively handle increased motion complexity.

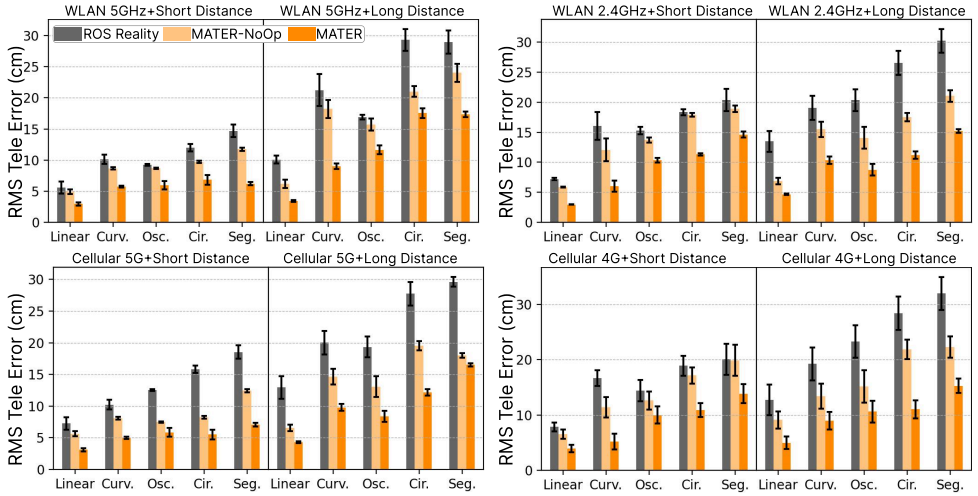


Fig. 12. Teleoperation Error comparison under Xavier-to-Nano

Simulation vs. Real Robot. We evaluate MATER in both simulation and on a physical robot to verify that improvements persist in practical deployment. In simulation run on PC-to-PC hardware (Fig. 10), MATER reduces RMS teleoperation error by up to 54.5% across network conditions and motion patterns. On a real robot in the PC-to-PC setup (Fig. 11), MATER still achieves up to 48.2% lower teleoperation error, proving that the robot trajectory coordination remains effective under real execution. The teleoperation error gap reflects the real-world tracking effects absent in simulation, such as actuation lag and tighter feasible speed/curvature limits. Together, these results show that MATER's improvements transfer to practical XR teleoperation scenarios.

Mission Completion Time. To evaluate mission completion time using pre-recorded motion patterns, we simulate user pauses by re-sending the last user pose whenever the teleoperation error in the 3D frame exceeds 5 cm. This is set by multiplying the user-perceivable M2M latency of 100 ms from §2.2 with the maximum robot manipulator speed of 50 cm/s in order to establish a lower bound for the pause time. Fig. 13 shows that under PC-to-PC hardware with WLAN 5GHz network, MATER reduces the mission completion time by up to 47.7% in short distance and 40.2% in long-distance experiments, due to the teleoperation error reduction achieved through the dual-reconstruction paradigm. MATER also decreases the standard deviation of mission completion time by up to 31.4%, owing to local synchronization that alleviates potential GPU contention. The reduced mission completion time indicates that user pauses are largely avoided in the controlled motion input, achieving a smooth and continuous teleoperation experience in these settings. Real-world teleoperation experience will be evaluated in §5.4.

5.3 Network Dynamics

Network Latency. We isolated the effect of network latency by minimizing fluctuations and packet drops. While keeping other system configurations unchanged, we connected the XR and robot host machine within the same network segment and controlled the network round-trip delay by adjusting traffic control utilities on both sides, maintaining evenly distributed latency. Fig. 14(A) demonstrates the RMS teleoperation error for ROS Reality and MATER as the round-trip network latency increases from 100 ms to 1000 ms, with the shaded area indicating standard deviation. The results show that ROS Reality experiences up to an 84.7% increase in RMS teleoperation

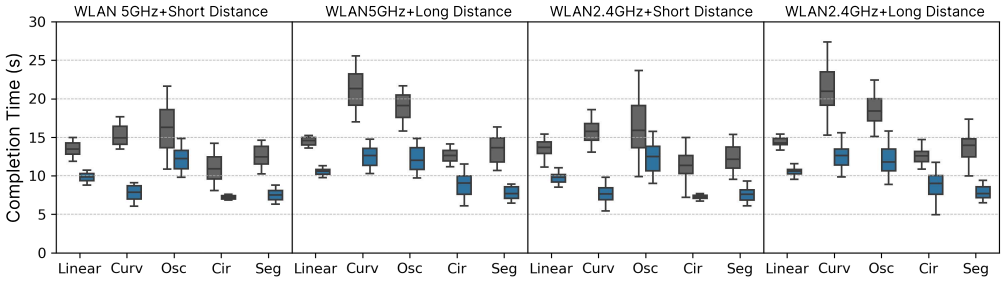
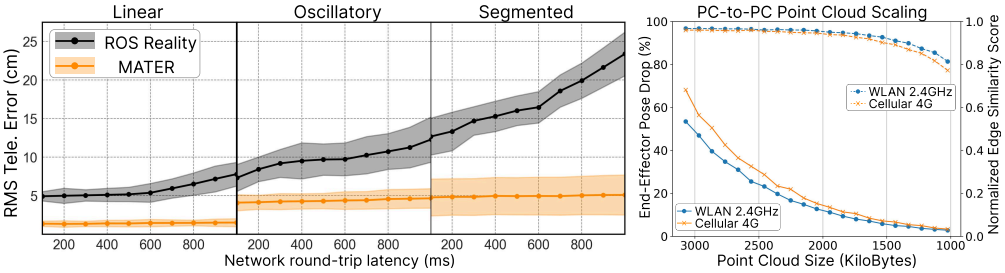


Fig. 13. Mission Completion time under PC-to-PC.



(A) Effect of Network Latency on Teleoperation Error **(B)** Point Cloud Scaling in Limited Bandwidth

Fig. 14. Effect of network latency and bandwidth.

error, whereas MATER incurs only a 14.7% increase. This discrepancy arises because, in ROS Reality, teleoperation error increases proportionally with network delay, due to the M2M latency enlarged by longer network latency. In contrast, MATER leverages dual-reconstruction to mitigate teleoperation errors induced by extended network latency, limiting the impact to the initial frames before the arrival of the first robot pose and gripper intensity in XR. Based on these results, we conclude that MATER experiences teleoperation error growth primarily due to degraded network fluctuations and limited bandwidth, as observed in each of Fig. 10, 11, 12. In simulation using the simplest linear motion, ROS Reality increases teleoperation error by 6.81 cm, whereas MATER increases it by only 3.59 cm, achieving a 47.3% reduction relative to ROS Reality. This improvement stems from MATER decoupling XR visualization and robot planning from network dependency, thereby enhancing system robustness under deteriorating network conditions.

Note that our network setup uses a cloud VPN server and is therefore not strictly symmetric, since the uplink and downlink paths between XR and robot can experience different latency and fluctuation in practice. In addition, increasing uplink/downlink asymmetry degrades MATER differently depending on the constrained direction: XR-to-robot degradation mainly affects reconstructed user motion on the robot side, while robot-to-XR degradation mainly affects reconstructed robot state and point cloud drops on the XR side.

Low Bandwidth Performance. We evaluated the performance of MATER under conditions where WLAN 2.4 GHz and Cellular 4G cannot support full information transmission. To this end, we conducted a parameter study of the point cloud scaling factor r and recorded the resulting point cloud size. Fig. 14(B) plots the tradeoff between the drop in robot poses and the quality of the point cloud, as measured by the edge similarity score [37]. We selected the edge similarity score because it captures the perceptual impact of point cloud modifications on human visual systems, and we

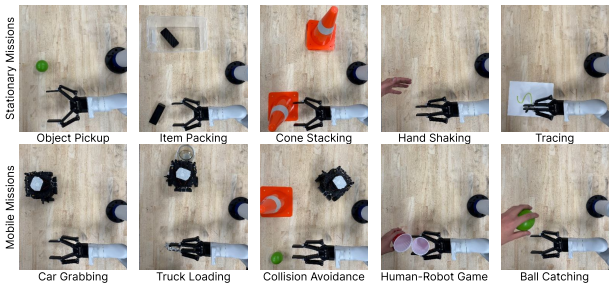


Fig. 15. Real-world mission settings.

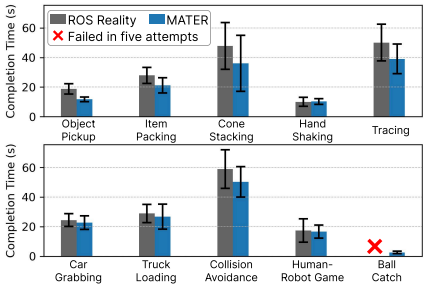


Fig. 16. Real-world completion time.

normalized the score between 0 and 1, where 1 indicates an identical point cloud and 0 indicates no matching points with the original. The results show that MATER reduces robot pose drops by up to 64.8% while incurring at most a 19.5% decrease in edge similarity. This improvement is achieved because bandwidth-adaptive point cloud scaling reduces overall transmission payload size while preserving structural integrity to minimize perceptual differences.

5.4 Real-world Case Study

Setup. To test performance in real-world applications, we designed 10 missions covering both stationary and mobile missions, each featuring a unique goal as shown in Fig. 15. We performed every mission five times using both ROS Reality and MATER, with rest periods between missions to mitigate XR-induced motion sickness. To minimize bias, we randomly alternated between ROS Reality and MATER without disclosing which framework was active during each trial. We conducted the experiment using Xavier-to-Nano on WLAN 2.4GHz network with a long distance setup.

Mission Completion Time. We measured the mission completion time for each mission and marked in red those missions that failed to complete in all five attempts. Fig. 16 illustrates the mission completion times, with error bars indicating standard deviation. In most scenarios, MATER reduced mission completion time by up to 37.7%. However, for human-in-the-loop missions such as Hand Shaking and Human-Robot Game, the improvement was only about 4.4% because human participation remained a significant factor in the total mission completion time. Notably, MATER enabled users to complete the rapid-motion ball-catching mission, owing to its mutually-aware capability, which delivers low teleoperation error, allowing users to view the projected robot motion before the remote robot pose is received.

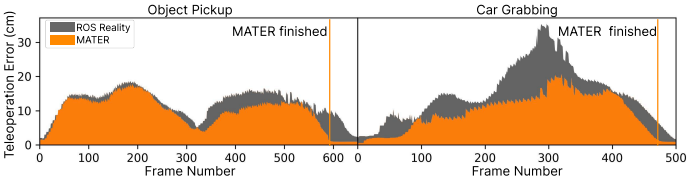


Fig. 17. Teleoperation error in real-world missions.

Teleoperation Quality. We profiled the teleoperation error during two trials that covered both stationary and mobile missions and plotted per-frame values in Fig. 17. In these trials, MATER reduced mission completion time by up to 10.1%, while maintaining on average 25.4% and 57.2% lower teleoperation error in the Object Pickup and Car Grabbing missions, respectively. This improvement occurs because, as the robot accelerates to catch up with a moving car, ROS Reality is unable to provide timely robot pose updates due to its network dependency, whereas MATER promptly reflects the robot motion by reconstructing the pose locally. This proves

a smooth teleoperation as both the mission completion time and teleoperation errors are effectively reduced during real-world missions.

6 Related Work

XR teleoperation System. Recent XR teleoperation researches have facilitated the use of various XR devices to control a wide range of robotic platforms, including robotic manipulators [3, 19, 48–51], mobile robots [2, 52, 53], drones [54], humanoid robots [7–10, 55], medical robots [4–6], and extraterrestrial robots [27, 31]. These systems allow users to interact with remote robots via two modes: *egocentric*, in which the robot imitates the user's real-time movements [3, 7–9], and *robocentric*, where user issues direct commands to robot's joints from a third-person view [18].

Research on system architectures for XR teleoperation is still in its infancy. Existing approaches adopt a design similar to ROS Reality [13, 29], where XR devices and robots exchange user control commands and robot pose updates for robot control and XR visualization via network communication. Consequently, system performance is highly dependent on network conditions, leaving it extremely vulnerable to any network delay, variation, and bandwidth limitation. While some studies have employed predictive displays [27, 31] to visualize the robot's near-future state to mitigate delays in seconds for space teleoperation missions, it is not applicable to real-time control scenarios that require the prediction to finish execution within each frame update. These prior approaches adopt only individual techniques and therefore lack the coordinated system-level design needed to jointly decouple XR visualization and robot execution from strict synchronization. In contrast, MATER introduces an architectural shift through bidirectional state reconstruction and coordination mechanisms.

Robot Control in XR Teleoperation. Unlike many conventional robot controls that operate with predetermined trajectories, XR teleoperation requires the robot to perform real-time planning and motion execution [11, 19, 36, 50, 51, 56]. Current methods directly map user poses from the XR interface to the robot and set joint velocities based on the latest available user and robot poses. However, this approach demands fine-grained and timely transmission of user data, making it subject to disruptions from network dynamics. While recent work has focused on enhancing pose mapping and joint control for various robot models to improve control accuracy [2, 10, 19, 48, 49], the errors introduced by network dynamics remain largely unaddressed.

7 Conclusion

We identify fundamental architectural drawbacks in the existing XR teleoperation frameworks and formulate two key challenges, C1 and C2, in §2.3. We address them with MATER, an end-to-end framework that combines mutually-aware state reconstruction (§3) with coordinated system components (§4). Our evaluation (§5) shows that MATER reduces both teleoperation error and mission completion time over state-of-the-art frameworks. MATER still faces limitations: it does not account for external factors on the robot side, and online scheduling could further improve robustness by adapting to execution-time variation across components.

Acknowledgment

This work was supported by the National Science Foundation (NSF) grants 1943265, 2312395, 2300525, 2312397, and 2343653.

References

[1] Vikash Kumar, Rutav Shah, Gaoyue Zhou, Vincent Moens, Vittorio Caggiano, Abhishek Gupta, and Aravind Rajeswaran. Robohive: A unified framework for robot learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[2] Abdeldjallil Naceri, Dario Mazzanti, Joao Bimbo, Domenico Prattichizzo, Darwin G Caldwell, Leonardo S Mattos, and Nikhil Deshpande. Towards a virtual reality interface for remote robotic teleoperation. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 284–289. IEEE, 2019.

[3] Ruishuo Xu, Weijun Wang, Wei Feng, Zhaokun Zhou, Boyoung An, Ruizhen Gao, and Kaichen Zhou. Design of a human-robot interaction system for robot teleoperation based on digital twinning. In *2022 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS)*, pages 720–726. IEEE, 2022.

[4] Intuitive. Da Vinci 5, 2025. https://www.intuitive.com/en-us/products-and-services/da-vinci/5/.

[5] Kateryna Zinchenko and Kai-Tai Song. Autonomous endoscope robot positioning using instrument segmentation with virtual reality visualization. *IEEE Access*, 9:72614–72623, 2021.

[6] David Black, Yas Oloumi Yazdi, Amir Hossein Hadi Hosseinabadi, and Septimiu Salcudean. Human teleoperation-a haptically enabled mixed reality system for teleultrasound. *Human–Computer Interaction*, 39(5-6):529–552, 2024.

[7] Xuxin Cheng, Jialong Li, Shiqi Yang, Ge Yang, and Xiaolong Wang. Open-television: Teleoperation with immersive active visual feedback. *arXiv preprint arXiv:2407.01512*, 2024.

[8] Aadhithya Iyer, Zhuoran Peng, Yinlong Dai, Irmak Guzey, Siddhant Haldar, Soumith Chintala, and Lerrel Pinto. Open teach: A versatile teleoperation system for robotic manipulation. *arXiv preprint arXiv:2403.07870*, 2024.

[9] Tairan He, Zhengyi Luo, Xialin He, Wenli Xiao, Chong Zhang, Weinan Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. Omnih2o: Universal and dexterous human-to-humanoid whole-body teleoperation and learning. *arXiv preprint arXiv:2406.08858*, 2024.

[10] Matthias Hirschmanner, Christiana Tsiourti, Timothy Patten, and Markus Vincze. Virtual reality teleoperation of a humanoid robot using markerless human upper body pose imitation. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 259–265. IEEE, 2019.

[11] Mohammad Bakhshalipour and Phillip B Gibbons. Agents of autonomy: A systematic study of robotics on modern hardware. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 7(3):1–31, 2023.

[12] Yunpeng Su, Leo Lloyd, Xiaoqi Chen, and J Geoffrey Chase. Latency mitigation using applied hmms for mixed reality-enhanced intuitive teleoperation in intelligent robotic welding. *The International Journal of Advanced Manufacturing Technology*, 126(5):2233–2248, 2023.

[13] David Whitney, Eric Rosen, Daniel Ullman, Elizabeth Phillips, and Stefanie Tellex. Ros reality: A virtual reality framework using consumer-grade hardware for ros-enabled robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[14] Minzhao Lyu, Rahul Dev Tripathi, and Vijay Sivaraman. Metavradar: Measuring metaverse virtual reality network activity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 7(3):1–29, 2023.

[15] Shihan Lin, Yi Zhou, Xiao Zhang, Todd Arnold, Ramesh Govindan, and Xiaowei Yang. Tiered cloud routing: Methodology, latency, and improvement. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 9(1):1–41, 2025.

[16] Sandeepa Bhuyan, Shulin Zhao, Ziyu Ying, Mahmut T Kandemir, and Chita R Das. End-to-end characterization of game streaming applications on mobile platforms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(1):1–25, 2022.

[17] Kinova Robotics. Kinova ros. https://github.com/Kinovarobotics/kinova-ros.

[18] Marco Gallipoli, Sara Buonocore, Mario Selvaggio, Giuseppe Andrea Fontanelli, Stanislao Grazioso, and Giuseppe Di Gironimo. A virtual reality-based dual-mode robot teleoperation architecture. *Robotica*, pages 1–24, 2024.

[19] Dinh Tung Le, Sheila Sutjipto, Yujun Lai, and Gavin Paul. Intuitive virtual reality based control of a real-world mobile manipulator. In *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 767–772. IEEE, 2020.

[20] David Whitney, Eric Rosen, Elizabeth Phillips, George Konidaris, and Stefanie Tellex. Comparing robot grasping teleoperation across desktop and virtual reality with ros reality. In *Robotics Research: The 18th International Symposium ISRR*, pages 335–350. Springer, 2019.

[21] Bryan R Galarza, Paulina Ayala, Santiago Manzano, and Marcelo V Garcia. Virtual reality teleoperation system for mobile robot manipulation. *Robotics*, 12(6):163, 2023.

[22] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 151–165, 2015.

[23] Suiping Zhou, Wentong Cai, Bu-Sung Lee, and Stephen J. Turner. Time-space consistency in large-scale distributed virtual environments. *ACM Transactions on Modeling and Computer Simulation*, 14(1):31–47, 2004.

[24] Lothar Pantel and Lars C. Wolf. On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st Workshop on Network and System Support for Games (NetGames)*, pages 79–84, 2002.

[25] Hassan Iqbal, Ayesha Khalid, and Muhammad Shahzad. Dissecting cloud gaming performance with decaf. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3), December 2021.
[26] Teemu Kämäräinen, Matti Siekkinen, Antti Ylä-Jääski, Wenxiao Zhang, and Pan Hui. A measurement study on achieving imperceptible latency in mobile cloud gaming. In *Proceedings of the 8th ACM Multimedia Systems Conference (MMSys)*, pages 88–99, 2017.
[27] Thomas B Sheridan. Space teleoperation through time delay: Review and prognosis. *IEEE Transactions on robotics and Automation*, 9(5):592–606, 1993.
[28] Henrikke Dybvik, Martin Løland, Achim Gerstenberg, Kristoffer Bjørnerud Slåttsveen, and Martin Steinert. A low-cost predictive display for teleoperation: Investigating effects on human performance and workload. *International Journal of Human-Computer Studies*, 145:102536, 2021.
[29] Siemens. ROS Sharp, 2024.
[30] J. M. P. van Waveren. The asynchronous time warp for virtual reality on consumer hardware. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, VRST '16, New York, NY, USA, 2016.
[31] Neil McHenry, Jason Spencer, Patrick Zhong, Jeremy Cox, Michael Amiscaray, KC Wong, and Gregory Chamitoff. Predictive xr telepresence for robotic operations in space. In *2021 IEEE Aerospace Conference (50100)*, pages 1–10. IEEE, 2021.
[32] Hem Regmi and Sanjib Sur. Argus: Predictable millimeter-wave picocells with vision and learning augmentation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(1):1–26, 2022.
[33] Jay H Lee and N Lawrence Ricker. Extended kalman filter based nonlinear model predictive control. *Industrial & Engineering Chemistry Research*, 33(6):1530–1541, 1994.
[34] Hind Taud and Jean-Franccois Mas. Multilayer perceptron (mlp). In *Geomatic approaches for modeling land change scenarios*, pages 451–455. Springer, 2017.
[35] Yu-Ping Wang, Wende Tan, Xu-Qiang Hu, Dinesh Manocha, and Shi-Min Hu. Tzc: Efficient inter-process communication for robotics middleware with partial serialization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7805–7812. IEEE, 2019.
[36] Torsten Kröger and Friedrich M Wahl. Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. *IEEE Transactions on Robotics*, 26(1):94–111, 2009.
[37] Zian Lu, Hailiang Huang, Huanqiang Zeng, Junhui Hou, and Kai-Kuang Ma. Point cloud quality assessment via 3d edge similarity measurement. *IEEE Signal Processing Letters*, 29:1804–1808, 2022.
[38] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.
[39] NorthStar. Northstar Next, 2023.
[40] Nvidia. Jetson AGX Xavier, 2018.
[41] Nvidia. Jetson Orin Nano, 2023.
[42] TP-Link. Archer C7 AC1750 Wireless Dual Band Gigabit Router, 2015.
[43] Samsung. Samsung Galaxy S23 128GB, 2023.
[44] Godot. Godot engine, August 2017. https://godotengine.org/.
[45] KhronosGroup. OpenXR Software Development Kit, 2024. https://github.com/KhronosGroup/OpenXR-SDK-Source.
[46] Elin. VR Gallery House, 2021. https://skfb.ly/6WXMx.
[47] Muhammad Huzaifa, Rishi Desai, Samuel Grayson, Xutao Jiang, Ying Jing, Jae Lee, Fang Lu, Yihan Pang, Joseph Ravichandran, Finn Sinclair, Boyuan Tian, Hengzhi Yuan, Jeffrey Zhang, and Sarita V. Adve. Illixr: Enabling end-to-end extended reality research. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pages 24–38, 2021.
[48] Dong Wei, Bidan Huang, and Qiang Li. Multi-view merging for robot teleoperation with virtual reality. *IEEE Robotics and Automation Letters*, 6(4):8537–8544, 2021.
[49] Bao Xi, Shuo Wang, Xuemei Ye, Yinghao Cai, Tao Lu, and Rui Wang. A robotic shared control teleoperation method based on learning from demonstrations. *International Journal of Advanced Robotic Systems*, 16(4):1729881419857428, 2019.
[50] Christian Just, Tobias Ortmaier, and Lueder A Kahrs. A user study on robot path planning inside a virtual reality environment. In *ISR 2018; 50th International Symposium on Robotics*, pages 1–6. VDE, 2018.
[51] Luis Pérez, Eduardo Diez, Rubén Usamentiaga, and Daniel F García. Industrial robot control and operator training using virtual reality interfaces. *Computers in Industry*, 109:114–120, 2019.
[52] Harvey Stedman, Basaran Bahadir Kocer, Mirko Kovac, and Vijay M Pawar. Vrtab-map: A configurable immersive teleoperation framework with online 3d reconstruction. In *2022 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 104–110. IEEE, 2022.
[53] Michael Walker, Zhaozhong Chen, Matthew Whitlock, David Blair, Danielle Albers Szafir, Christoffer Heckman, and Daniel Szafir. A mixed reality supervision and telepresence interface for outdoor field robotics. In *2021 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), pages 2345–2352. IEEE, 2021.

[54] Anurag Sai Vempati, Harshit Khurana, Vojtech Kabelka, Simon Flueckiger, Roland Siegwart, and Paul Beardsley. A virtual reality interface for an autonomous spray painting uav. *IEEE Robotics and Automation Letters*, 4(3):2870–2877, 2019.

[55] Konstantinos Theofilis, Jason Orlosky, Yukie Nagai, and Kiyoshi Kiyokawa. Panoramic view reconstruction for stereoscopic teleoperation of a humanoid robot. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 242–248. IEEE, 2016.

[56] Kaveh Kamali, Ilian A Bonev, and Christian Desrosiers. Real-time motion planning for robotic teleoperation using dynamic-goal deep reinforcement learning. In *2020 17th Conference on Computer and Robot Vision (CRV)*, pages 182–189. IEEE, 2020.

Received January 2026; revised March 2026; accepted April 2026